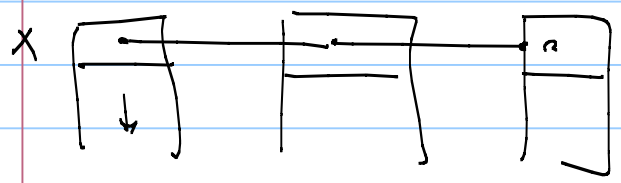
Lecture 14

Note Title 2/26/2008




```
private static ThreadLocal<Foo> fooHolder = new
                    ThreadLoc<Foo>(){
                        public Foo initialValue() = new Foo(); }

public static getFoo() { fooHolder.get()}
```

Immutable Objects –

- state cannot change after construction
- all fields must be final
- must be properly constructed

```
public static H h;
public void init(){ h = new H(42);}
```

```
public class H {
  private int n;
  public H (int n)
    {this.n = n;}
  public void check ()
  { if (n!=n) {...};
  }
}
```

Publishing rules:

Immutable objects: don't have to worry

Mutable objects (properly constructed):

idea – the reference to the object and to its state become available to other threads at the same time.

- initialize in a static initializer

  public static H h = new H(42);
- store the reference in a volatile field or AtomicReference
- store reference in a final field
- store ref. in a field consistently guarded by a lock. ✓

E. I.

Effectively immutable objects —

- if published in one of the above way E. I. objects require no further synchronization

Atomic classes

AtomicInteger – generalization of volatile

.get()

.set(v)

increment, decrement, add, addAdd

CompareAndSet ⟵

```
private value;
Synchronized int CASwap (int expected, int newV) {
        int oldV = value;
        if (oldV == expected) value = newV;
        return (oldV);
}
boolean CAset (int expected, int newV){
      return (expected == CASwap(expected, newV));
}
```

atomic increment — value is an AtomicInteger

```
int v;
do { v = value.get(); } while (v != value.CASwap(v, v+1));
```

Exam

Declarative : power and limitations of decl. concurrency
including limitations on includable language features.
Reading Oz code
Writing simple Oz code

Message passing : programs patterned as port objects;
Erlang use of higher order constructs to encapsulate
non-functional behaviors

Java - consequences of the memory model for concurrent programming.

- examples of the monitor pattern